

## Structured query language query join optimization by using rademacher averages and mapreduce algorithms

Yathish Aradhya Bandur Chandrashekariah<sup>1</sup>, Dinesha H. A.<sup>2</sup>

<sup>1</sup>Department of Computer Science and Engineering, VTU-RRC, Belagavi, India

<sup>2</sup>Department of Computer Science and Engineering, Shridevi Institute of Engineering and Technology, Tumkur, India

### Article Info

#### Article history:

Received May 30, 2023

Revised Sep 11, 2023

Accepted Sep 28, 2023

#### Keywords:

Join query optimization  
Lehigh university benchmark  
Rademacher averages  
Structured query language  
Waterloo sparql diversity test suite

### ABSTRACT

Query optimization involves identifying and implementing the most effective and efficient methods and strategies to enhance the performance of queries. This is achieved by intelligently utilizing system resources and considering various performance metrics. Table joining optimization involves optimizing the process of combining two or more tables within a database. Structured query language (SQL) optimization is the progress of utilizing SQL queries in the possible way to achieve fast and accurate database results. SQL optimization is critical to decreasing the no of queries in research description framework (RDF) and the time for processing a huge number of relatable data. In this paper, four new algorithms are proposed such as hash-join, sort-merge, rademacher averages and mapreduce for the progress of SQL query join optimization. The proposed model is evaluated and tested using waterloo sparql diversity test suite (WatDiv) and lehigh university benchmark (LUBM) benchmark datasets in terms of time execution. The results represented that the proposed method achieved an enhanced performance of less execution time for various queries such as Q3 of 5362, Q8 of 5921, Q9 of 5854 and Q10 of 5691 milliseconds. The proposed gives better performance than other existing methods like hybrid database-map reduction system (AQUA+) and join query processing (JQPro).

*This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.*



### Corresponding Author:

Yathish Aradhya Bandur Chandrashekariah  
Department of Computer Science and Engineering, VTU-RRC  
Belagavi, India  
Email: docs.kit@gmail.com

## 1. INTRODUCTION

Database management involves a significant focus on query optimization, which plays a crucial role in determining the most efficient order of query operators and their respective implementation algorithms [1]. Join is a sufficient operator and can have implemented in various ways on the size of its inputs and outputs [2]. The query optimizer of a database management system is used for selecting a query plan which can be executed effectively [3]. In the join query, the time of execution is mainly depending upon how every join is physically executed [4]. Analytical queries are the popular queries but analytical queries take more time to complete the process. Multi-join collects the data from various data sources or tables to provide the material for applications like decision support, data sharing and integration [5], [6]. In big data analyzing and aggregating data from various data sources for decision-making and prediction in different fields like scientific research, industry, business and public utilities can yield highly valued services and products [7]. The effectiveness of multi-join query can affect various factors like the count of joins and the count of data,

join selectivity, order of join execution, data storage location, resources, strategies and methods that parallel optimization used [8].

The structured query language (SQL) is created to manipulate, store and query relational databases [9], [10]. SQL is carried out by a leading database management system data base management system (DBMS) and used in a wide variety of sectors [11]. A DBMS contains irrelevant information and programs for manipulating the stored data. The information is separately entered into the tables, records and attributes [12]. SQL allows a huge amount of information for processing and the ability to resolve combinatorial optimization problems that include clustering, and informetric data processing [13]. One of the major issues in query optimization and plan execution is the choosing of the order to process the join operation among the tables [14], [15]. In the recent years, the semantic data capacity is maximized with amount of research description framework (RDF) datasets excessive of trillion triples in RDF archives. So, RDF dataset's size grows simultaneously [16]. Although with an increasing amount of RDF triples, the complex multi-RDF queries become major demand [17], [18]. Certain times those complex queries generate some general sub-expressions in one or multi queries processing as a batch. It is also critical to decrease the amount of RDF queries and execution time for huge number of relatable data in distributed environment [19], [20].

The existing works on query-based optimization techniques are listed as: Ji *et al.* [21] introduced a double deep q network (DDQN) method for join-order query optimization. In this method, the selected actions were done by using a dynamic progressive search strategy which can evaluate the information gain (IG) for the uncertainty of data in the network that can step up the learning speed and attain effective potential exploration. In this model, if increase the number of joins, the delay time of the model was lower. The training of the model was delayed due to the elimination of errors to some extent in the cost model. Mohsin *et al.* [22] developed a quantum inspired ant colony algorithm (QIACO) model for join query cost optimization in distributed databases. The search method was employed on the search space, utilizing the information obtained from the category tables in the database, to enhance the join order. The advantage of using the quantum method was it solved major issues quickly and more efficiently. The model does not consider the mutation and modified crossover process. Pavlopoulou *et al.* [23] suggested a runtime dynamic optimization for join queries in big data management systems. In the model, at the time of query execution place many re-optimization points and at each re-optimization point considered only the next cheapest join and do not form the whole plan and search among all the possible join ordering variations. The advantage of the model w on querying big data estimated the small error in big datasets which improves query optimization. For predicates, the model has no selectivity estimation and for choosing the right join orders and join algorithm opportunities were missed. Renukadevi *et al.* [24] developed a moth flame and tunicate swarm algorithm (MF-TSA) for solving query optimization and optimizing the query platforms in the crowd sourcing platform. In this model, to encrypt the data the homomorphic encryption method is used. Two types of crowd-controlled administrators are used in query design such as crowd powered selection (CSelect) and crowd powered join (CJoin) were joined for query processing. The developed query optimization model was used to reduce latency and cost. The MF-TSA query optimizer used integrating tables for selecting the cost of join and it was processed only once. Pang *et al.* [25] implemented an hybrid database-map reduction system (AQUA+) model for query optimization in a hybrid database-map reduction system. In this model, the distributed database was used for maintaining the data, and on top of the same set of cluster nodes map reduce engines were deployed. AQUA+ and SQL-like queries are divided into two parts. One was submitted for processing by the database engine and the other part reads the information from the database into the map-reduce engine for query optimization. The model used novel tuning for query optimization to improve the performance. Due to a insufficient map-reduce engine, the sharding and filtering performance gets worse when the cluster gets larger.

From the overall analysis, the existing methods has the limitations such as high execution time, the model has no selectivity estimation and for selecting right join orders and join algorithm opportunities were missed, used integrating tables for selecting the cost of join, it was processed only once and there is an insufficient map-reduce engine, the sharding and filtering performance gets worse when the cluster gets larger. To address these complications, the paper proposed a join query processing method for big RDF data, by utilizing the mapreduce framework proposed four new algorithms such as hash-join, sort-merge, rademacher averages and mapreduce -join for SQL join query processing. A rademacher averages is calculated which measures the complexity of queries and reduces the complexity of queries which reduces the execution time of query optimization. The major contribution of this research is listed as follows:

- A novel model for joint optimization of SQL query is proposed by using the rademacher averages and mapreduce algorithms. Four new algorithms are developed by adopting the mapreduce framework such as hash-join, sort-merge, rademacher average and mapreduce join for the progress of SQL query join optimization.
- A novel methodology is proposed for the calculation of rademacher averages bound of all given large datasets. Rademacher bound measures the upper limit of input size of training data set and making sure the trained model is within the probably approximate correct framework (PAC). Rademacher averages are used to the measure complexity of a queries which reduces the execution time of query optimization.

---

*Structured query language query join optimization by using ... (Yathish Aradhya Bandur Chandrashekariah)*

- The two benchmarks lehigh university benchmark (LUBM) and waterloo sparql diversity test suite (WatDiv) v06 are utilized for evaluation of SQL join query optimization.

The remaining part of this research is as: a detailed explanation of the proposed methodology is given in section 2. The results and discussion of this proposed method are illustrated in section 3. The conclusion of this paper is presented in section 4.

## 2. PROPOSED METHODOLOGY

In the proposed methodology develop a join optimization for SQL query using hash join, sort-merge, rademacher averages, and mapreduce framework. The methodology consists of data source, preprocessing, and mapreduce framework, which is shown in Figure 1. The data are updated into HIVE, and from the storage, the input selector takes the information in triple formats. Based on the query to minimize the time it uses mapreduce to find the relevant results and the plan generation is used to minimize the quantity of data and execution time of the query [26]. The hadoop cluster is used to retrieve the information depending on the machine chosen by the hadoop cluster. Hash join is the first algorithm; all the selected inputs are added to the hash table by utilizing a join algorithm. The hash join algorithm outcome is utilized in a sort-merge algorithm which is the second algorithm that is utilized to identify the defined values in the join features for every part of the information that has the similarity of all datasets. The output of the sort-merge algorithm is used in rademacher averages which is the third algorithm used to sort the query for optimization. The collaboration of the list of join features is utilized in mapreduce join algorithm to identify the chosen information utilizing the mapreduce technique.

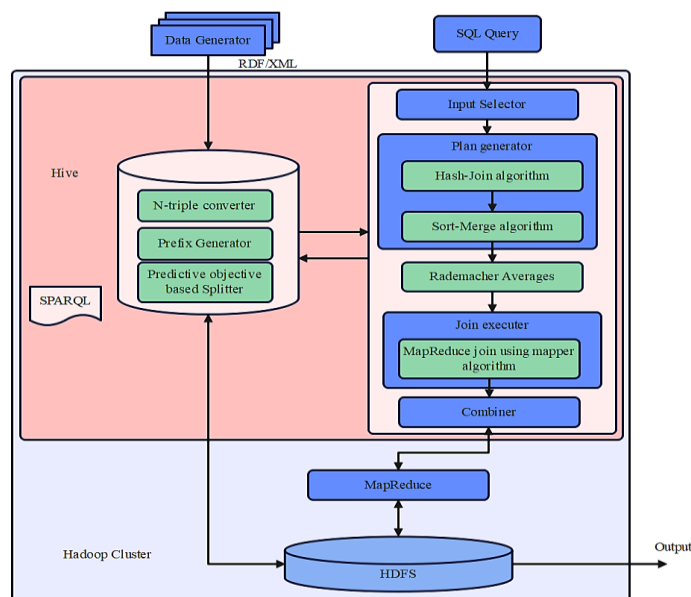


Figure 1. Workflow of proposed methodology

### 2.1. Dataset

The join optimization for SQL query consists of two major components such as data resource and mapreduce technique. The datasets used are benchmark datasets such as LUBM. The component of the data source converts huge URL data into RDF formation. The data pre-processor uses an N-triples converter module for converting *RDF* into *N – triples* serialization. The *N – triple* file of the *RDF* graph is an input data resource that can be updated as an SQL query. The user sends the query through the SQL query associate and then the query is generated by the mapreduce technique.

### 2.2. Hash join algorithm

The hash-join algorithm is mainly created for the input selection module. In the building phase, inside the table, the first set of data is loaded with the table [27]. In the probe stage, the huge dataset is joined and scanned with the related triple utilized in it. In the first algorithm, the table partitions are described as P and Q in WatDiv and LUBM datasets. The algorithm arranges both datasets into join qualities and then

combines the arranged datasets and then groups datasets with similar scores in a column of join. The hash-join algorithm joins various datasets effectively if they are arranged in various databases. The hash join algorithm for query optimization follows the given (1):

$$R \bowtie_{r(a)\theta s(b)} S \quad (1)$$

The algorithm for performing (1) is as:

```

for each tuple s in S do
{hash on join attributes s(b)
  Place tuples in a hash table based on hash values};
for each tuple r do
{hash on join attributes r(a)
  If r hashes to a nonempty bucket of hash table for S
  Then {if r θ – matches any s in bucket
    Concatenate r and s
    Place in relation Q}};
```

In the above algorithm, a similar hashing function should be utilized for both relations. In the probe stage, the huge dataset is joined and scanned with the related triple utilized in it. The hash-join algorithm joins various datasets effectively if they are arranged in various databases. The hash join algorithm is used for the input selector module; the selected inputs are sent to sort-merge join algorithm.

### 2.3. Sort-merge join algorithm

The sort-merge algorithm verifies if the plan generation time is shorter than the query processing time, and proceeds to execute the next query [28]. The processing of query and plan generation of the RDF datasets are the same and these elements are updated to the  $P$  and  $Q$  datasets. If the  $p$  and  $q$  are equal the progress will terminate. By covering a particular key range, the merge learns from two various parts of reducing outcomes. The sorting progress groups every tuple with similar importance in the join column. The separation is utilized by trying to differentiate  $P$  tuples in a separate with  $Q$  tuples in a similar separation, thus further ignoring the enumeration of  $P$  and  $Q$ . Sort-merge join requires arranging the tables depending on the general key and joining the arranged tables by examining them in order. It may be utilized in conditions where a huge number of data and performance is taken. The sort-merge algorithm for query optimization and plan generation follows the given (2):

$$R \bowtie_{r(a)=s(b)} S \quad (2)$$

The algorithm for performing the sort-merge method is;

```

sort R on r(a);
sort S on s(b);
read first tuple from R;
read first tuple from S;
for each tuple r do
{while s(b) < r(a)
  Write s into Q
  Pad R – attributes with null values
  Read next tuple from S;
  If r(a) = S(b) then
    join r and s
  place in output relation Q};
```

### 2.4. Rademacher averages

Rademacher averages are an important method to learn the rate of coverage of a group of sample means to the expectations [29]. A novel methodology is proposed for the calculation of rademacher averages bound of all given large datasets. Rademacher bound measures the upper limit of input size of the training data set and making sure the trained model is within the probably PAC. rademacher averages are used to the measure complexity of a queries which reduces the execution time of query optimization. The rademacher complexity is described on an arbitrary calculation part, the sample part contains a limited domain  $D$  and the specific distribution over the particles of  $D$ . Let  $F$  be the functions from  $D$  to the interval  $[0,1]^2$  and let  $S = \{s_1, \dots, s_l\}$  be a group of  $l$  individual uniform samples from  $D$ . For each  $f \in F$ , describe as (3). Given  $S$ , the highest divergence of  $m_S(f)$  from  $m_D(f)$  between every  $f \in F$ , i.e., the quantity describe as (5). For  $1 \leq i \leq l$ , let  $\lambda_i$  be a variable of rademacher random ( $r.v.$ ), i.e., a  $r.v$  which takes value 1 with possibility  $\frac{1}{2}$  and -1 with possibility  $\frac{1}{2}$ . The  $r.v.$ 's  $\lambda_i$  are individual describe as (6).

$$m_D(f) = \frac{1}{|D|} \sum_{c \in D} f(c) \text{ and } m_S(f) = \frac{1}{l} \sum_{i=1}^l f(s_i) \quad (3)$$

It holds, describe as (4):

$$m_D(f) = E[f] \text{ and } E[m_S(f)] = m_D(f) \quad (4)$$

$$\sup_{f \in F} |m_S(f) - m_D(f)| \quad (5)$$

$$R_F(S) = E_\lambda \left[ \sup_{f \in F} 1/l \sum_{i=1}^l \lambda_i f(s_i) \right], \quad (6)$$

Where the expectations are only taken the rademacher  $r.v.$ 's, that is conditioning on  $S$ . The quantity  $R_F(S)$  is defined as the rademacher average of  $F$  on  $S$ . The relation between  $R_F(S)$  and the highest divergence is the main output in the theory of statistical studying. Let  $S$  be a group of  $l$  individual uniform samples from  $D$ . Let  $n \in (0,1)$ . Then with the possibility of at least  $1 - \eta$ , describe as (7):

$$\sup_{f \in F} |m_S(f) - m_D(f)| \leq 2R_F(S) + \frac{\ln \frac{3}{\eta} + \sqrt{\left( \ln \frac{3}{\eta} + 4lR_F(S) \right) \ln \frac{3}{\eta}}}{l} + \sqrt{\frac{\ln \frac{3}{\eta}}{2l}} \quad (7)$$

Computing the expectation in *w.r.t* the rademacher  $r.v.$ 's is not straightforward and can be computationally expensive, requiring a time-consuming monte carlo simulation. For the above reason, higher bounds to the rademacher average are commonly estimated in the space of  $R_F(S)$ . A strong and effective-to-execute bound is implemented in the theorem. Given  $S$  taken for every  $f \in F$ , the vector  $v_f, S = (f(s_1), \dots, f(s_l))$  and let  $V_S = \{v_f, S, f \in F\}$  be the group of those vectors ( $|V_S| \leq |F|$ , as there can be distinct functions of  $F$  with similar vectors). Let  $w: R^+ \rightarrow R^+$  be the function,  $w(r)$  describe as (8):

$$w(r) = \frac{1}{r} \ln \sum_{v \in V_S} \exp \frac{r^2 \|v\|_2^2}{2l^2} \quad (8)$$

where  $\| \cdot \|_2$  denoted the  $l_2$  - norm (euclidean norm). Then:

$$R_F(S) \leq \min_{r \in R^+} w(r) \quad (9)$$

The function  $w$  is convex, continuous in  $R^+$ , and has first and second derivatives  $r$  everywhere in its domain, so it is possible to minimize it effectively by using standard convex optimization methods. The euclidean norm is described by utilizing (9). Rademacher algorithm and statistical learning can be assumed that a learning algorithm chooses its hypothesis from certain fixed hypothesis class  $H$ , generalization of error analysis gives theoretical results bounding generalization of error of hypothesis  $h \in H$  which will be based on the sample and properties of hypothesis class. For any hypothesis  $h$ , their generalization error is the possibility that a randomly drawn example is wrongly classified and describe as (10), any learning algorithm aims to identify a hypothesis with a less generalization error, it cannot compute on a sample per se, as it would also depend on probability distribution  $P$ . However, an attempt can be made to approximate the generalization error of hypothesis  $h$ , by their errors in training on  $n$  examples,  $\varepsilon_n(h)$  is describe as (11).

$$\varepsilon_p(h) = P(h(x) \neq y) \quad (10)$$

$$\varepsilon_n(h) = 1/n \sum_{i=1}^n L(h(x_i), y_i) \quad (11)$$

where  $L$  is 0/1 loss function

$$L(z, z') = \{(1, \text{if } z \neq z'), (0, \text{otherwise})\}$$

Empirical risk minimization (ERM) is a principle according to which only such hypothesis whose training error is minimal is to be chosen. To ensure that ERM yields a hypothesis with less generalization error the bound  $\sup_{h \in H} |\varepsilon_p(h) - \varepsilon_n(h)|$  can be used. The difference of training error of hypothesis  $h$  on  $n$  examples and true generalization error converges to 0 in probability as  $n$  tends to infinity, provided examples are independent and identical in distribution while the hypothesis class  $H$  is not complex. The rademacher random variable takes values  $+1$  and  $-1$  with the possibility of  $1/2$  for all. Let rademacher random variables  $r_1, r_2, r_3, \dots, r_n$  be  $(x_1, y_1), \dots, (x_n, y_n)$ . The rademacher penalty of hypothesis class  $H$  in query is described as (12) and by symmetrization inequality of the theory of empirical processes described as (13).

$$R_n(H) = \sup_{h \in H} |1/n \sum_{i=1}^n r_i L(h(x_i), y_i)| \quad (12)$$

$$E\{sup_{h \in H} |\varepsilon_p(h) - \varepsilon_n(h)|\} \leq 2E\{R_n(H)\}$$

$$\varepsilon_p(h) \leq \varepsilon_n(h) + 2R_n(H) + n(\delta, n) \quad (13)$$

Where  $n(\delta, n) = 5 \sqrt{(ln(\frac{2}{\delta}))/2n}$  is a less error term that takes care of fluctuations of examined random variables around the expectations. The benefits of the inequality (2) are raised from the truth which is their right-hand side based on the training sample and not on  $p$  directly, which means that  $R_n(H)$  may be computed with the algorithm utilized for ERM. Rademacher penalties can be applied to provide an approximate solution to the progressive sampling algorithm, particularly in the estimation of stopping time in terms of the data-dependent upper bound ( $\varepsilon$ -approximate) of the sample set used for training the algorithm. The minimal number of samples required to guarantee ERM is within a distance of  $\varepsilon$  from the generalization error of  $h$  for every  $h \in H$ ; in query describe as (14). The rademacher stopping time  $v(\varepsilon, \delta)$  with parameters  $(\varepsilon, \delta)$  for a hypothesis of class  $H$  in query is describe as (15).

$$Argmin\{P_r\{sup_{h \in H} |\varepsilon_p(h) - \varepsilon_n(h)| \geq \varepsilon\} \leq \delta \quad (14)$$

$$v(\varepsilon, \delta) = min\{n_i = 2^i n_0(\varepsilon, \delta) | R_{ni}(H) < \varepsilon\} \quad (15)$$

The rademacher average is utilized to attain the data-dependent upper bounds on the learnability of function classes, a function class with lesser rademacher averages is easy to study. The minimal number of samples required to guarantee ERM is within a distance of  $\varepsilon$  from the generalization error of  $h$  for every  $h \in H$ ; in query. Rademacher averages sorted only the selected query, these selected queries are used in mapreduce join algorithm.

## 2.5. Mapreduce join

The mapreduce join algorithm erases every non-joining variable which is irrelevant to the  $Q$  query.  $Q = \{X, Y, VZ, XY, XZ\}$ , where  $X, Y, VZ, XY$  and  $XZ$  show a group of attributes with their combination for query  $Q$  in the processing and erases the attribute non-merging  $V$  produces the  $Q$  outcomes =  $\{X, Y, Z, XY, XZ\}$ . *Job* is commonly developed as the storage space for every join operation where every task processed is stored respectively. It is significant to point out that ' $j$ ' shows the identity of the present task. To properly experiment with the join optimization method with RDF data, assume that when running locally in a real application, different large data applications vary dynamically and consider the cluster size used in the analysis. This experiment assumed that users may scale high or low work traces to their own needs. The benchmarks are applied to calculate the time execution performance, use of *CPU* and proposed to join query algorithms performance. Mapreduce progress is calculated by their timely execution. Various some another factors like the number of mapping challenges, the underlying network, the common jumbled data pattern and size mainly affect the process of mapreduce. Input and output types of mapreduce job can be described as (16):

$$(Input)k_1, v_1 \rightarrow map \rightarrow k_2, v_2 \rightarrow reduce \rightarrow k_3, v_3 (Output) \quad (16)$$

Where  $k_1, k_2, k_3$  are keys and  $v_1, v_2, v_3$  are values

## 3. EXPERIMENTAL RESULTS

The experimental result is given in this section. The section contains the experimental setup, benchmark dataset used, the effectiveness of the benchmark dataset, descriptions of datasets utilized, and comparative analysis of existing and proposed models are represented in detail. In this paper, the proposed rademacher averages are simulated using a python environment with the system requirements; RAM: 16GB, processor: Intel Core i7 and operating system: Windows 10 (64 bit).

### 3.1. Experimental setup

A hadoop cluster is used in the conducting experiment. In a distinct (VM), every node is installed. Set up a network connection between 5 computers where every computer has a consecrate ethernet cable attached to a switch. Then downloaded the hadoop software on every device to merge them into a single cluster. Separating workloads across various nodes can increase the progressing power and performance of clusters and allows quick progress of huge datasets or difficult calculations. The hadoop cluster consists of a master VM, namenode and various VM which work depending on nodemanager and datanode. Datanode is situated in a cluster by the namenode where an RDF query is kept. When the data are situated inside the

cluster, the nodemanager started the processing of data by utilizing the mapreduce which starts with input selector, maps, joins, jumbles, and decreases. The significance of the clustering progress may be seen in their capacity to minimize the number of information required for working. An algorithm can delete the unwanted computations and join operations by grouping the related data and resulting in quicker query time processing and minimized computational overhead. Batch progress may also process by a clustering process that is majorly utilized for handling large numbers of data. An algorithm may work batch progressing on data subsets by clustering similar data together that minimizes the overall performance time needed to solve all queries. The experiments are worked with 5 VMs on the linux ubuntu 19 hadoop cluster. Namenode and resourcemanager are utilized by the VMs when the datanode and manager are processed by others. Every VM has a 2.50GHz processor, 8GB main memory and 100GB of disk place configuration. A benchmark group of CPU and 10-intensive applications is used in the hadoop circulation like WatDiv and LUBM is utilized for process determination for evaluating the mapreduce jobs efficiently.

### 3.2. Dataset benchmarks

In this paper, two benchmarks are utilized such as LUBM and WatDiv v06 benchmarks. LUBM: The LUBM is created as a normal to completely ease the performance of semantic web archives. The main use of the benchmark is to examine the performance of the archives with coherent queries over a huge dataset committed to one realistic ontology. The benchmark includes the ontology of the domain of the university, customized and quotable processed information, a group of test queries and different performance metrics. LUBM has maximized to have a globally adopted RDF and OWLdataset benchmark around a data size of 1 GB.

WatDiv: the benchmark is created to calculate how an RDF data management system works on a huge number of sparql queries with various attributes and choosy attributes. WatDiv data giver allows users to develop their datasets along a specified language. So, users may maintain which attributes are updated in the dataset. How each attribute is good-designed and how various attributes are combined with others and the possibility that a kind  $X$  attribute is combined with a kind  $Y$  attribute and the multiplicity of those combinations. The test data for *WatDiv* is structured by utilizing the attributes. It is probable to give test datasets of different sizes by performing the data generator with various scale factors.

The execution ranges from 10 million to 1 billion RDF triples produced by utilizing the WatDiv data producer with scale factors of 10,100 and 1000 on 3 datasets. WatDiv is a much-maintained environment for RDF data management systems with much various underlying works than another benchmark, giving the producer all the various formats of queries. The motive for selecting the above-mentioned benchmarks in the research is both benchmarks of the datasets are the much often utilized to perform RDF datasets as baseline benchmarks for mapreduce.

### 3.3. Effectiveness of join optimization of WatDiv benchmark

In this section, Table 1 and Figure 2 represent the outcomes of join optimization on the WatDiv benchmark and calculation of time execution for several queries of different size 10M, 100M, and 1000M using the WatDiv benchmark. The queries described in Table 1 are *complex* (1 – 3), *linear* (1 – 5), *sflake* (1 – 5), and *star* (1 – 2). Table 1 observed that each query has various execution time, certain queries contain a huge time execution and other contains a low time execution. The join optimization of time execution 10M with the WatDiv benchmark dataset is processed efficiently. The query execution time of complex 1 is maximized continuously. The execution time is minimized for complex 1 query with the huge power of the CPU. This examination is difficult as the process of the proposed method is not in a state across different query sizes in the WatDiv benchmark. Although, the developed method is executed effectively in this research. The whole outcomes represent that along with a high amount of triples, the timely execution of certain classes is minimized slowly. This outcome represents the utilizing join optimization method to execute an RDF query that can decrease the time that is taken to solve a task in comparison to the other previous general query that gives constant times for every outcome.

### 3.4. Effectiveness of join optimization of LUBM benchmark

In this section, Table 2 and Figure 3 represent the results of join optimization on the LUBM benchmark and the calculation of time execution for a number of queries of different size 500 M, 2000 M, and 1000 M using the LUBM benchmark.

In the Table 2, the number of query parts included from one to 14 queries are selected. The outcomes represent that these queries work well than common RDF queries. Fourteen queries in Table 2 are processed to calculate the time execution and it represents that some query's time execution is maximized. Table 2 represents the overall mean time processing for queries of size 500M. From the result, noticed that the query  $Q_{11}$  has the maximum time execution. Query  $Q_4, Q_3, Q_5, Q_{14}$ , and  $Q_6$  are the good-executing query on a 500M query size. The outcome acquired in this part with the evaluation on the LUBM benchmark

is better, with no higher than the 6.6 s of time execution in all queries. These values are significant to sorting the outcome acquired in this section on the LUBM benchmark.

Table 1. Calculation of execution time for several queries using WatDiv

	Execution time 10 M	Execution time 100 M	Execution time 1000 M
Complex 1	5,530.00	5,590.00	17,000.00
Complex 2	5,600.00	6,520.00	5,900.00
Complex 3	5,560.00	5,550.00	5,600.00
Linear 1	6,000.00	5,540.00	6,000.00
Linear 2	5,570.00	6,000.00	5,550.00
Linear 3	5,985.00	5,570.00	5,500.00
Linear 4	5,650.00	6,568.00	5,620.00
Linear 5	5,520.00	5,575.00	5,630.00
Sflake 1	5,725.00	5,577.00	6,700.00
Sflake 2	5,525.00	5,585.00	5,550.00
Sflake 3	5,830.00	5,555.00	5,530.00
Sflake 4	5,565.00	5,570.00	5,535.00
Sflake 5	5,550.00	5,590.00	5,540.00
Star 1	6,570.00	5,580.00	5,700.00
Star 2	5,575.00	7,500.00	5,600.00
Star 3	5,578.00	5,650.00	5,535.00
Star 4	5,590.00	5,600.00	5,550.00
Star 5	5,780.00	5,550.00	5,537.00
Star 6	5,585.00	5,550.00	5,600.00
Star 7	5,587.00	6,400.00	5,530.00

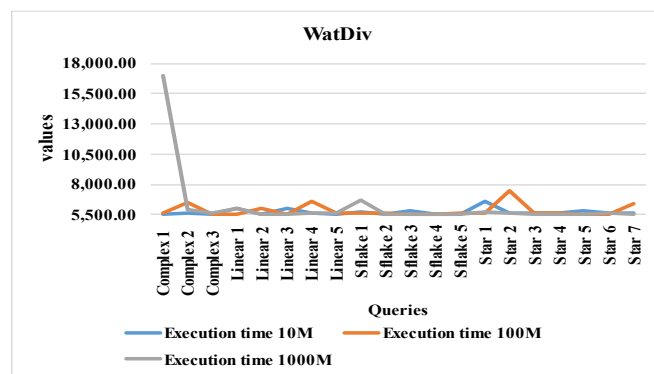


Figure 2. Calculation of execution time for an amount of the queries using WatDiv

Table 2. Calculation of time execution for an amount of the queries using LUBM

	Execution time (ms) 500 M	Execution time (ms) 2000 M	Execution time (ms) 1000 M
Q1	5,975.00	5,940.00	5,800.00
Q10	5,990.00	5,940.00	6,150.00
Q11	7,450.00	5,850.00	6,070.00
Q12	6,180.00	5,600.00	6,000.00
Q13	6,200.00	6,070.00	5,890.00
Q14	5,880.00	5,880.00	5,810.00
Q2	5,895.00	5,940.00	5,600.00
Q3	5,800.00	5,745.00	6,000.00
Q4	5,820.00	5,990.00	6,150.00
Q5	5,850.00	5,905.00	5,860.00
Q6	5,900.00	6,095.00	6,000.00
Q7	6,000.00	5,885.00	5,815.00
Q8	6,090.00	6,010.00	6,170.00
Q9	5,960.00	5,950.00	5,990.00

### 3.5. Comparative analysis

The section shows the comparative analysis of proposed method with existing methods like AQUA+ [25] and join query processing (JQPro) [30] in terms of execution time. Two alternative cases are examined while analyzing the proposed method. In case 1, the execution time of query is taken in seconds where as in case 2, the execution time of query is taken in milliseconds. Table 3, the proposed method is compared to AQUA+ [25] for



case 1. Table 4, shows the comparative analysis of proposed method with JQPro [30] for case. From the results of Tables 3 and 4, it is clear that the proposed method shows enhanced performance than existing methods.

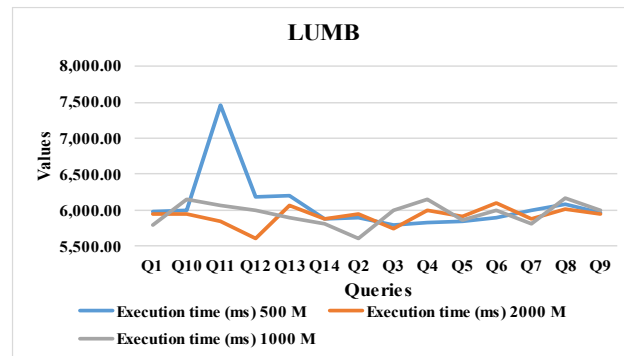


Figure 3. Calculation of time execution for the number of queries using LUMB

Table 3. Comparative analysis for case 1

Author	Method	Execution time for various queries (ms)			
		Q3	Q8	Q9	Q10
Elzein <i>et al.</i> [30]	JQPro	5,957.00	6,264.00	6,081.00	6,054.00
Proposed method	Rademacher averages	5,362.00	5,921.00	5,854.00	5,691.00

Table 4. Comparative analysis for case 2

Author	Method	Execution time for various queries (sec)			
		Q3	Q8	Q9	Q10
Pang <i>et al.</i> [25]	AQUA+	180.00	220.00	500.00	210.00
Proposed method	Rademacher averages	5,362.00	5,921.00	5,854.00	5,691.00

### 3.6. Discussion

This section provides the discussion about proposed method and its result comparisons. The AQUA+ [25] has limitation such as due to an insufficient map-reduce engine, the sharding and filtering performance gets worse when the cluster gets larger. The existing model JQPro [30] has limitations such as the semantic data capacity is maximized with the amount of RDF datasets excessive of trillion triples in RDF repositories. So, the RDF dataset's size grows simultaneously. Although with an increasing amount of RDF triples, the complex multi-RDF queries become major demand. Certain times those complex queries generate some general sub-expressions in single or multiple queries running as a batch. It is also critical to decrease the amount of RDF queries and execution time for a huge number of relatable data in distributed environment. To address these complications, the paper proposed a join query processing model for big RDF data, by utilizing the mapreduce framework proposed four new algorithms such as hash-join, sort-merge, rademacher averages and mapreduce - join for SQL join query processing. Hash join is utilized when the estimation of the combined tables is not already sorted on the combined columns. Merge sort simultaneously cuts down a list into various sublists until everyone has a single component then combines that sublists into a sorted list. Rademacher averages are used to the measure complexity of a class. Mapreduce joins algorithms by using a hadoop cluster which takes away the complexity of distributed programming. The computation result shows that the execution time of multiple tasks in parallel system is minimized. The proposed method achieved an enhanced performance of less execution time for various queries such as Q3 of 5,362.00, Q8 of 5,921.00, Q9 of 5,854.00, and Q10 of 5,691.00 milliseconds which is comparatively better performance than other existing methods like AQUA+ has execution time of Q3, Q8, Q9, and Q10 was 180, 220, 500, and 210 seconds respectively and JQPro has execution time of Q3, Q8, Q9, and Q10 was 5957, 6264, 6081, and 6054 milliseconds respectively.

## 4. CONCLUSION

In this research, a new method is proposed using rademacher averages for the SQL query join optimization. The developed model comprises four various join algorithms such as hash-join, sort-merge, rademacher averages and mapreduce join algorithms for effective optimization. Hash join is utilized when the estimation of the combined tables is not already sorted on the combined columns. Merge sort simultaneously cuts down a list into various sublists until everyone has a single component then combines

that sublists into a sorted list. Rademacher averages are used to the measure complexity of a class. Mapreduce joins algorithms by using a hadoop cluster which takes away the complexity of distributed programming. From the performance analysis it is concluded that the proposed model has performed well and tested using WatDiv and LUBM benchmark dataset, the results show that the proposed method achieved an enhanced performance of less execution time for various queries such as Q3 of 5,362.00, Q8 of 5,921.00, Q9 of 5,854.00, and Q10 of 5,691.00 milliseconds.

The proposed method has limitations such as limited range of query operations, dependence on mapreduce algorithm and tested on LUBM benchmark. Advanced query capabilities are needed in some scenarios and other distributed computing algorithms like apache spark or flink may give better scalability and performance. The future scope for the research is utilize the machine learning algorithms to the optimization of join query in distributed systems, machine learning algorithm like reinforcement learning to choose join algorithm, keys and partition of data for effective processing.




## REFERENCES

- [1] J. Torres-Jimenez, N. Rangel-Valdez, M. De-La-torre, and H. Avila-George, "An Approach to Aid Decision-Making by Solving Complex Optimization Problems Using SQL Queries," *Applied Sciences (Switzerland)*, vol. 12, no. 9, p. 4569, Apr. 2022, doi: 10.3390/app12094569.
- [2] R. B. Guo and K. Daudjee, "Research challenges in deep reinforcement learning-based join query optimization," in *Proceedings of the 3rd International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM 2020*, New York, NY, USA: ACM, Jun. 2020, pp. 1–6, doi: 10.1145/3401071.3401657.
- [3] D. Kumar and V. K. Jha, "An improved query optimization process in big data using ACO-GA algorithm and HDFS map reduce technique," *Distributed and Parallel Databases*, vol. 39, no. 1, pp. 79–96, Mar. 2021, doi: 10.1007/s10619-020-07285-z.
- [4] P. Michiardi, D. Carra, and S. Migliorini, "Cache-Based Multi-Query Optimization for Data-Intensive Scalable Computing Frameworks," *Information Systems Frontiers*, vol. 23, no. 1, pp. 35–51, Feb. 2021, doi: 10.1007/s10796-020-09995-2.
- [5] R. Mancini, S. Karthik, B. Chandra, V. Mageirakos, and A. Ailamaki, "Efficient Massively Parallel Join Optimization for Large Queries," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, Jun. 2022, pp. 122–135, doi: 10.1145/3514221.3517871.
- [6] X. X. Hu, J. Q. Xi, and D. Y. Tang, "Optimization for Multi-Join Queries on the GPU," *IEEE Access*, vol. 8, pp. 118380–118395, 2020, doi: 10.1109/ACCESS.2020.3002610.
- [7] B. Zheng, X. Li, Z. Tian, and L. Meng, "Optimization Method for Distributed Database Query Based on an Adaptive Double Entropy Genetic Algorithm," *IEEE Access*, vol. 10, pp. 4640–4648, 2022, doi: 10.1109/ACCESS.2022.3141589.
- [8] D. Sharkova et al., "Adaptive SQL Query Optimization in Distributed Stream Processing: A Preliminary Study," in *Communications in Computer and Information Science*, 2022, pp. 96–109, doi: 10.1007/978-3-030-93849-9\_7.
- [9] M. Ramadan, A. El-Kilany, H. M. O. Mokhtar, and I. Sobh, "RL\_QOptimizer: A Reinforcement Learning Based Query Optimizer," *IEEE Access*, vol. 10, pp. 70502–70515, 2022, doi: 10.1109/ACCESS.2022.3187102.
- [10] X. Li, H. Yu, L. Yuan, and X. Qin, "Query Optimization for Distributed Spatio-Temporal Sensing Data Processing," *Sensors*, vol. 22, no. 5, p. 1748, Feb. 2022, doi: 10.3390/s22051748.
- [11] M. Sharma, G. Singh, and R. Singh, "Clinical decision support system query optimizer using hybrid Firefly and controlled Genetic Algorithm," *Journal of King Saud University - Computer and Information Sciences*, vol. 33, no. 7, pp. 798–809, Sep. 2021, doi: 10.1016/j.jksuci.2018.06.007.
- [12] R. Soundararajan, S. R. Kumar, N. Gayathri, and F. Al-Turjman, "Skyline Query Optimization for Preferable Product Selection and Recommendation System," *Wireless Personal Communications*, vol. 117, no. 4, pp. 3091–3108, Apr. 2021, doi: 10.1007/s11277-020-07592-9.
- [13] M. S. Rafiq, X. Jianshe, M. Arif, and P. Barra, "Intelligent query optimization and course recommendation during online lectures in E-learning system," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 11, pp. 10375–10394, Nov. 2021, doi: 10.1007/s12652-020-02834-x.
- [14] A. S. Yadav and D. S. Kushwaha, "Query optimization in a blockchain-based land registry management system," *Ingenierie des Systemes d'Information*, vol. 26, no. 1, pp. 13–21, Feb. 2021, doi: 10.18280/isi.260102.
- [15] T. Alyas, A. Alzahrani, Y. Alsaawy, K. Alissa, Q. Abbas, and N. Tabassum, "Query Optimization Framework for Graph Database in Cloud Dew Environment," *Computers, Materials and Continua*, vol. 74, no. 1, pp. 2317–2330, 2023, doi: 10.32604/cmc.2023.032454.
- [16] Y. Choubik, A. Mahmoudi, M. M. Himmi, and L. El Moudnib, "STA/LTA trigger algorithm implementation on a seismological dataset using hadoop mapreduce," *IAES International Journal of Artificial Intelligence*, vol. 9, no. 2, pp. 269–275, Jun. 2020, doi: 10.11591/ijai.v9.i2.pp269-275.
- [17] S. W. Kareem, R. Z. Yousif, and S. M. J. Abdalwahid, "An approach for enhancing data confidentiality in hadoop," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 20, no. 3, pp. 1547–1555, Dec. 2020, doi: 10.11591/ijeecs.v20.i3.pp1547-1555.
- [18] M. M. Khudhair, A. Al-Rammahi, and F. Rabee, "An innovative fractal architecture model for implementing MapReduce in an open multiprocessing parallel environment," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 30, no. 2, pp. 1059–1067, May 2023, doi: 10.11591/ijeecs.v30.i2.pp1059-1067.
- [19] I. Sassi, S. Anter, and A. Bekkhoucha, "A spark-based parallel distributed posterior decoding algorithm for big data hidden markov models decoding problem," *IAES International Journal of Artificial Intelligence*, vol. 10, no. 3, pp. 789–800, Sep. 2021, doi: 10.11591/ijai.v10.i3.pp789-800.
- [20] J. M. Arockiam and A. C. S. Pushpanathan, "MapReduce-iterative support vector machine classifier: novel fraud detection systems in healthcare insurance industry," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 1, pp. 756–769, Feb. 2023, doi: 10.11591/ijece.v13i1.pp756-769.
- [21] L. Ji, R. Zhao, Y. Dang, J. Liu, and H. Zhang, "Query Join Order Optimization Method Based on Dynamic Double Deep Q-Network," *Electronics (Switzerland)*, vol. 12, no. 6, p. 1504, Mar. 2023, doi: 10.3390/electronics12061504.
- [22] S. A. Mohsin, S. M. Darwish, and A. Younes, "QIACO: A Quantum Dynamic Cost Ant System for Query Optimization in Distributed Database," *IEEE Access*, vol. 9, pp. 15833–15846, 2021, doi: 10.1109/ACCESS.2021.3049544.




- [23] C. Pavlopoulou, M. J. Carey, and V. J. Tsotras, "Revisiting Runtime Dynamic Optimization for Join Queries in Big Data Management Systems," *Advances in Database Technology - EDBT*, vol. 25, pp. 1–12, 2022, doi: 10.5441/002/edbt.2022.01.
- [24] M. Renukadevi, E. A. M. Anita, and D. M. Geetha, "An efficient privacy-preserving model based on OMFTSA for query optimization in crowdsourcing," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 24, Dec. 2021, doi: 10.1002/cpe.6447.
- [25] Z. Pang, S. Wu, H. Huang, Z. Hong, and Y. Xie, "AQUA+: Query Optimization for Hybrid Database-MapReduce System," *Knowledge and Information Systems*, vol. 63, no. 4, pp. 905–938, Feb. 2021, doi: 10.1007/s10115-020-01542-4.
- [26] X. Ji, M. Zhao, M. Zhai, and Q. Wu, "Query Execution Optimization in Spark SQL," *Scientific Programming*, vol. 2020, pp. 1–12, Feb. 2020, doi: 10.1155/2020/6364752.
- [27] D. Bilidas and M. Koubarakis, "In-memory parallelization of join queries over large ontological hierarchies," *Distributed and Parallel Databases*, vol. 39, no. 3, pp. 545–582, Sep. 2021, doi: 10.1007/s10619-020-07305-y.
- [28] A. Modi *et al.*, "New Query Optimization Techniques in the Spark Engine of Azure Synapse," *Proceedings of the VLDB Endowment*, vol. 15, no. 4, pp. 936–948, Dec. 2021, doi: 10.14778/3503585.3503601.
- [29] L. Pellegrina, C. Cousins, F. Vandin, and M. Riondato, "MCRapper: Monte-Carlo Rademacher Averages for Poset Families and Approximate Pattern Mining," *ACM Transactions on Knowledge Discovery from Data*, vol. 16, no. 6, pp. 1–29, Dec. 2022, doi: 10.1145/3532187.
- [30] N. M. Elzein, M. A. Majid, I. A. T. Hashem, A. O. Ibrahim, A. W. Abulfaraj, and F. Binzagr, "JQPro:Join Query Processing in a Distributed System for Big RDF Data Using the Hash-Merge Join Technique," *Mathematics*, vol. 11, no. 5, p. 1275, Mar. 2023, doi: 10.3390/math11051275.

## BIOGRAPHIES OF AUTHORS



**Yathish Aradhya Bandur Chandrashekariah**    is a research scholar at VTU Belagavi with 12 years of teaching experience and 2 year of industry experience. He has obtained MTech Degree from RVCE, Bangalore. He has published 6 international journals and 5 conference articles. He has executed various projects in computer science and currently working as assistant professor at Kalpataru Institute of Technology, Tiptur in India. He has conducted programming courses online. He has presented technical talks and published study materials. He can be contacted at email: docs.kit@gmail.com.



**Dr. Dinesha H. A. (Professor (CSE) and Dean (R&D))**    is a serving as Professor and Dean (R&D), at Shridevi Institute of Engineering and Technology. He is a fledgling idealistic entrepreneur Indulged passionately in serving and uplifting society through advance research, innovation, and knowledge imparting in the arena of Engineering and Technology. He is the founder of an IT industry named Cyber sena(R&D) India Private Limited, India in the domain of Cybersecurity and forensic investigations. He is completed a Post-Doctoral Fellow in the Cyber forensic domain. He has completed Ph.D. in Cloud Computing Security, Post-Graduation M.Tech in Software Engineering. He has proved his excellence & surpassed the field of virtualization technologies during his tenure at VMware (R&D) India Private Limited. He is credited with meticulous publication of 75+ research papers in national and international research platforms. He is awarded with 15+ Indian and international patents for his innovative work. He has delivered outstanding 100+ keynotes and technical sessions which had amazing impacts among the audience. He has also proudly worked on many innovative projects in DRDO, Pune, Govt of India, PESIT Bangalore, KCTU, and Elevate-21 Govt. of Karnataka. He has taken efforts in executing Rs. 2+ crores funded projects on server virtualizations, mail servers, UTM Firewall deployment, digital forensics, virtual classroom, etc. As an academic researcher he takes immense interest in guiding and supervising research scholars. He has been guiding and supervising Ph.D. scholars in their research and also in securing their doctoral degree. As of now, 5 research scholars have been awarded for Ph.D. under his supervision. At present he is supervising 7 Ph.D. research scholars. He had steered 25+ P.G and 50+ U.G students' projects in the various domains of Computer Science and Engineering. Few of the projects got recognized turn out to be as a solution for the real time problems to the Ministry of Railways, Ministry of Food Processing, Ministry of Information & Broadcasting (GOI), and Ministry of Defence, Govt. of India. As an acknowledgement, he has received many awards such as InSc Young Achiever and Best researcher award-2020, Innovative Start-up and Entrepreneur, Global Teaching Excellence Award-2021, and many best research papers awards from 2009 to 2022. He can be contacted at email: sridini@gmail.com.